A thick black L-shaped frame surrounds the text. It starts at the top left, goes right, then down, then right again at the bottom right.

MACHINE LEARNING FOR SOFTWARE ENGINEERING

Ingegneria del Software 2 – Di Totto Luca 0333084

Sommario

1. Introduzione
2. Obiettivo
3. Metodologia:
 1. *Recupero delle metriche*
 2. *Costruzione del dataset*
 3. *Tecniche di utilizzo*
4. Risultati:
 1. *BookKeeper*
 2. *Syncope*
5. Conclusioni
6. Considerazioni sulla validità dei risultati
7. Link

Introduzione

- Il testing nei progetti software è un'attività che richiede tempo e risorse economiche significative. Effettuare test esaustivi sull'intero progetto risulta spesso oneroso e di difficile applicazione.
- Ridurre questi costi e tempi diventa quindi fondamentale per ottimizzare le risorse disponibili. Una possibile soluzione consiste nell'utilizzare dati storici per prevedere quali classi potrebbero presentare bug in futuro.

Obiettivo

- L'obiettivo è rendere il processo di testing più mirato ed efficiente. A tal fine, nel progetto di studio:
 - *Sono stati applicati diversi modelli di Machine Learning per prevedere la bugginess delle classi nei progetti analizzati.*
 - *È stata condotta un'analisi dei risultati ottenuti, variando le tecniche di addestramento.*
- Lo scopo finale è individuare la configurazione ottimale per i due progetti Apache esaminati: BookKeeper e Syncope.

Metodologia

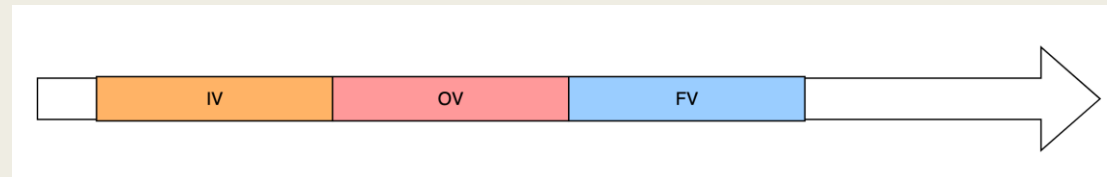
- Le metriche vengono raccolte dai seguenti progetti:
 - *Apache BookKeeper*
 - *Apache Syncope*
- Successivamente, vengono costruiti i dataset di training e testing, sui quali vengono applicati algoritmi di Machine Learning per eseguire le predizioni. In particolare:
 - Sono stati utilizzati tre modelli: *Random Forest, Naive Bayes e IBK*.
 - Sono state adottate tre tecniche: *Feature Selection, Sampling (Over e Under) e Cost Sensitivity*.

Metodologia – Recupero metriche

- Si introduce il concetto di "**ciclo di vita**" del bug, in cui una classe viene considerata affetta se presente tra la **Injected Version (IV)** e la **Fixed Version (FV)**.

$$IV \leq \text{BUGGY} < FV$$

- La **timeline** completa dell'arco temporale di vita del bug è la seguente:



- Nello specifico:
 - *IV: # della release dalla quale il bug viene introdotto;*
 - *OV: # della release dalla quale il bug viene rilevato;*
 - *FV: # della release nella quale il bug viene risolto e chiuso.*

Metodologia – recupero metriche

- Le informazioni relative a **IV (Injected Version)**, **OV (Opening Version)** e **FV (Fixed Version)** vengono recuperate dalle issue sulla piattaforma **JIRA**.
 - **Problema:** non tutte le versioni su **JIRA** contengono dati sulla **IV**.
- Per ovviare a questa mancanza, si utilizza la **tecnica di proportion**, che permette di stimare la **IV** nei ticket privi di questa informazione. Il calcolo si basa su un parametro di proporzionalità **p**, derivato dai ticket che includono la **IV**.

$$p = \frac{FV - IV}{FV - OV}, \text{ se } FV=OV \text{ allora } p = \frac{FV - IV}{1}$$

$$IV = \max\{1, FV - (FV - OV) * p\}$$

Metodologia – recupero metriche

- Nel progetto vengono adottate due modalità di calcolo della **proportion**:
 - **Cold Start**: utilizzata quando il numero di ticket disponibili è insufficiente per calcolare una proportion affidabile (inferiore alla dimensione della finestra). In questo caso, il valore viene stimato utilizzando ticket provenienti da progetti correlati.
 - **Moving Window**: calcola dinamicamente la proportion basandosi esclusivamente su un numero limitato di ticket recenti, determinato dalla dimensione della finestra.

Metodologia – Costruzione del dataset

- Per ridurre il fenomeno dello **snoring**, ovvero lo sbilanciamento dovuto alla diminuzione delle classi buggy nelle release più recenti, viene eliminata la metà più recente delle release.
- Per ciascuna coppia (release – classe) vengono recuperate delle **metriche** (illustrate nella slide successiva) per permettere ai classificatori di effettuare le predizioni.
- Per ciascuna coppia (release – classe) viene inserita una label (come ultima colonna del dataset) che indica se la stessa era affetta da **bug** o meno, nella release specifica.

Metodologia – Costruzione del dataset

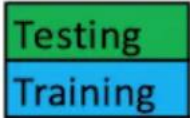
LOC	size (lines of code)
LOC TOUCHED	somma delle LOC aggiunte e cancellate
NUMBER OF REVISIONS	numero delle revisioni sulla classe
LOC ADDED	LOC aggiunte
AVG LOC ADDED	media delle LOC aggiunte per revisione
NUM OF AUTHORS	numero di autori
MAX LOC ADDED	massimo delle LOC aggiunte sulle revisioni
TOTAL LOC REMOVED	LOC totali rimosse sulle revisioni
MAX LOC REMOVED	massimo delle LOC rimosse sulle revisioni
AVG LOC TOUCHED	media delle LOC aggiunte e cancellate sulle revisioni
BUGGY	classe buggy o non buggy

- Le metriche **intra-release** (non cumulative) raccolte per ogni classe sono mostrate nella figura di fianco.
- La metrica **BUGGY** viene calcolata nel seguente modo:
 - Vengono estratti da **JIRA** tutti i ticket di tipo **"bug"** con risoluzione **"fixed"** e stato **"closed"** o **"resolved"**.
 - Le classi modificate da un commit associato a uno dei ticket precedentemente estratti vengono etichettate come **buggy** per tutte le **AV** (ovvero le versioni comprese tra **IV** e **FV**).

Metodologia – Costruzione del dataset

- Per effettuare le predizioni, i classificatori devono essere addestrati utilizzando il **training set**.
- Per raccogliere le statistiche dei classificatori, le predizioni devono essere verificate attraverso il **testing set**.
- La costruzione del **training set** e del **testing set** avviene tramite la tecnica di **Walk Forward**, una modalità **time-series** iterativa che segue i seguenti passaggi:
 - *Il **training set** viene costruito utilizzando le prime n release, con il labeling eseguito esclusivamente con le informazioni disponibili fino a quel momento.*
 - *Il **testing set** viene costruito, ad ogni iterazione, utilizzando le informazioni presenti nella " $n+1$ "-esima release, su cui vengono fatte le predizioni. Il labeling per questa release si basa su tutte le informazioni disponibili fino a quel punto.*

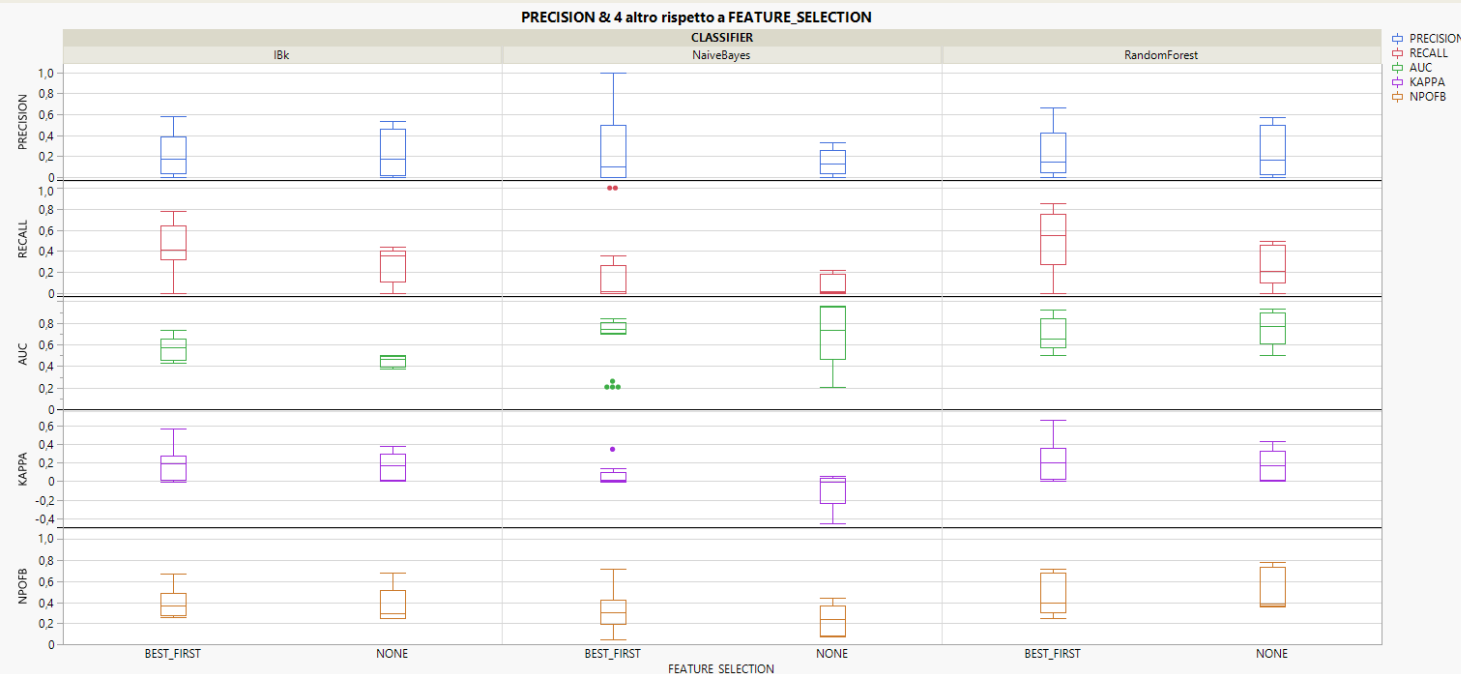
Run	Part				
	1	2	3	4	5
1	Training				
2	Training	Testing			
3	Training	Training	Testing		
4	Training	Training	Training	Testing	
5	Training	Training	Training	Training	Testing



Metodologia – Tecniche di utilizzo

- Vengono utilizzati i seguenti classificatori e le seguenti configurazioni:
 - ***Classificatori:***
 - Random Forest;
 - Naive Bayes;
 - IBK.
 - ***Configurazioni:***
 - Nessun filtro presente;
 - Feature selection (Best First);
 - Feature selection (Best First) + Balancing (Under Sampling);
 - Feature selection (Best First) + Balancing (Over Sampling);
 - Feature selection (Best First) + Sensitive learning (CFN = 10*CFP).

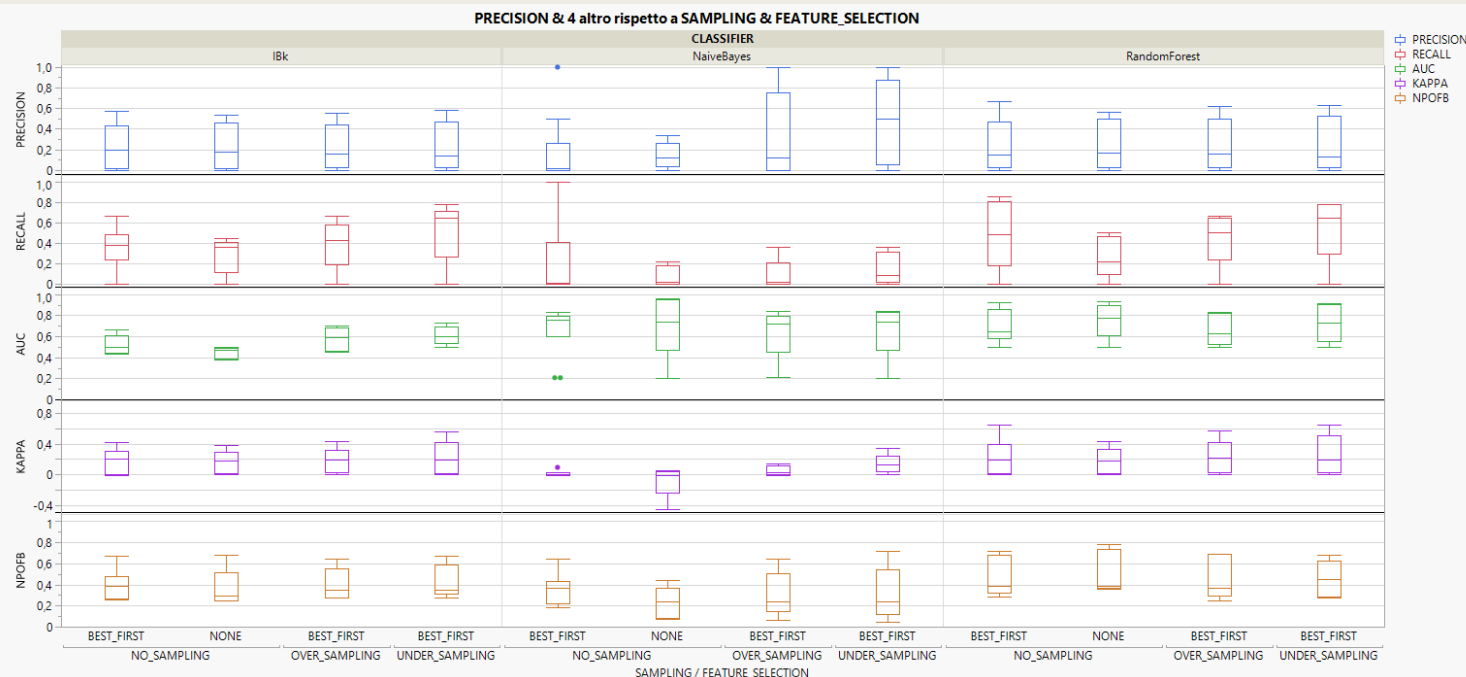
Risultati – Bookkeeper - esecuzione con feature selection



- Confrontando i risultati ottenuti in caso di utilizzo di feature selection con l'esecuzione senza nessun filtro, si nota che:
 - IBk:** valore di precision che resta pressochè invariato, mentre si nota che la recall è complessivamente migliorata, anche se presenta risultati molto variabili in quanto i 'baffi' del box plot sono più lunghi.
 - Naive Bayes:** in questo caso la precision rimane mediamente invariata, anche se in caso di utilizzo di feature selection presenta una variabilità più elevata. Stesse considerazioni per il valore di recall, che rimangono pressochè invariati.
 - Random Forest:** in questo caso è possibile notare una leggera riduzione del valore di precision, mentre il valore di recall è mediamente più elevato rispetto all'uso senza filtri.

In generale, possiamo considerare **Random Forest** leggermente migliore rispetto a **IBk**, mentre **Naive Bayes** risulta essere il peggiore tra quelli analizzati.

Risultati – Bookkeeper - esecuzione con feature selection e sampling

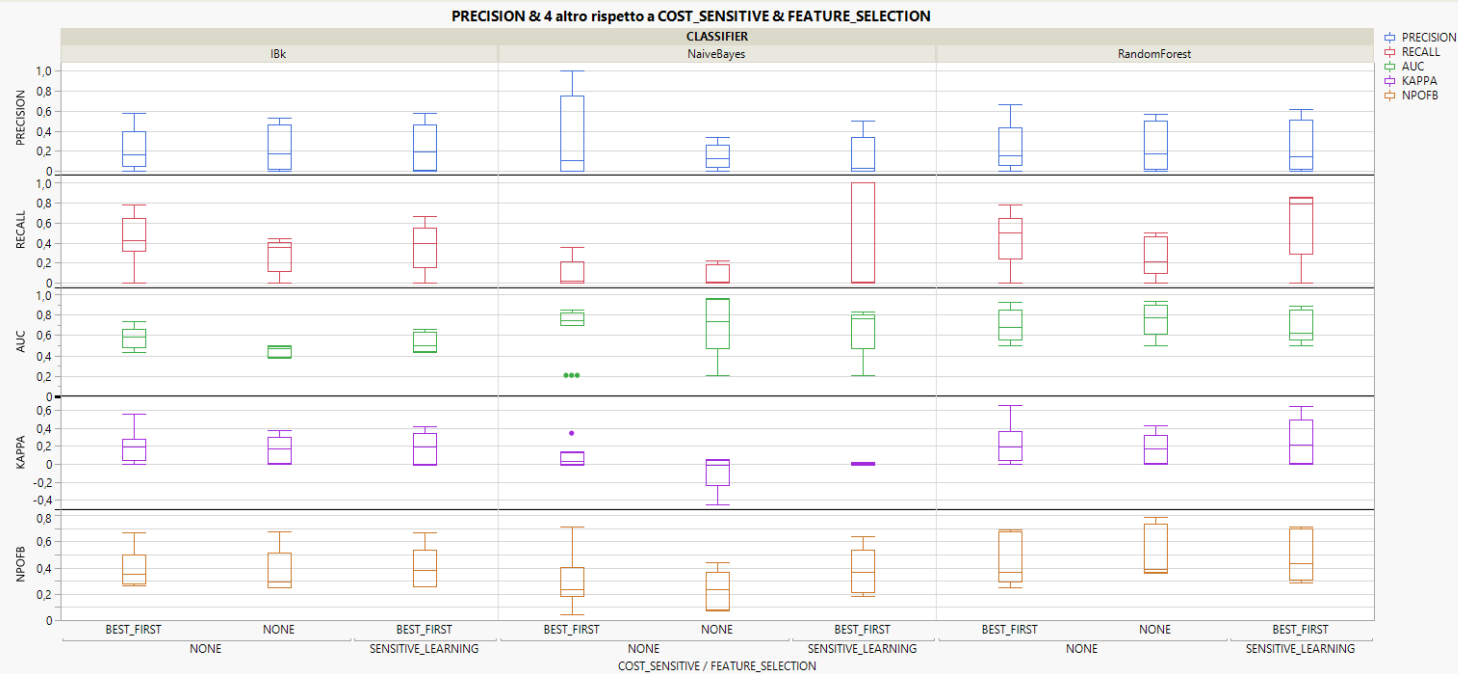


- Nelle esecuzioni con **Feature Selection** e **Over Sampling** troviamo:
 - **IBk**: valore di precision simile al caso senza filtraggio, mentre si nota un miglioramento più accentuato della recall dovuto all'aumento della percentuale di positivi nel training set
 - **Naive Bayes**: risulta essere ancora il classificatore peggiore in quanto i valori medi di precision e recall rimangono sostanzialmente invariati;
 - **Random Forest**: miglioramento notevole della recall rispetto alle esecuzioni senza filtraggio, leggero rispetto al caso con solo filtraggio. Precision che rimane invariata rispetto ai casi precedenti.

- Nelle esecuzioni con **Feature Selection** e **Under Sampling** si nota che:
 - **IBk**: notevole miglioramento del valore di recall rispetto a tutti i casi precedentemente analizzati, dovuto alla riduzione dei negativi all'interno del training set. Precision che invece rimane invariata.
 - **Naive Bayes**: valor medio di precision nettamente migliorato e superiore a tutti gli altri casi, recall leggermente superiore agli altri casi con stesso classificatore, ma notevolmente bassa se confrontata con altri classificatori.
 - **Random Forest**: precision generalmente costante rispetto al caso con solo filtraggio, mentre valore di recall molto migliorato.In tutti i casi con under sampling comunque la recall aumenta, come atteso

In generale, in caso di **Over Sampling**, **Random Forest** e **IBk** riportato risultati simili; in caso di **Under Sampling**, **IBk** e **Random Forest** hanno comportamenti simili con un netto miglioramento del valore di recall.

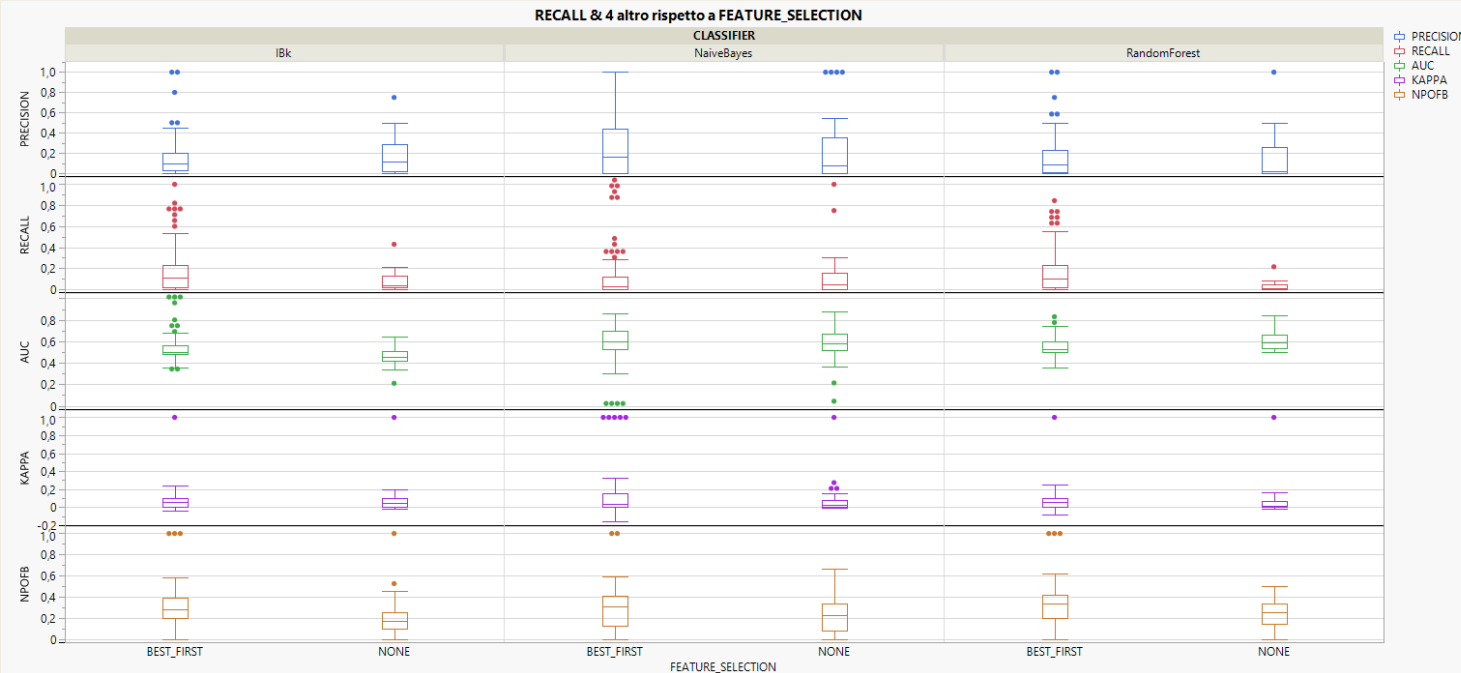
Risultati – Bookkeeper - esecuzione con feature selection e sensitive learning



- Nelle esecuzioni con **Feature Selection** e **Sensitive Learning** troviamo:
 - **IBk**: valore di precision costante e valore di recall simile e leggermente peggiore rispetto al caso con solo filtraggio
 - **Naive Bayes**: un valor medio di recall molto bassa anche se con elevata variabilità. Valore di precision leggermente inferiore rispetto agli altri casi
 - **Random Forest**: valore di precision simile agli altri due esempi, valore di recall nettamente migliorato

Random Forest risulta essere il classificatore migliore subito seguito da IBk.

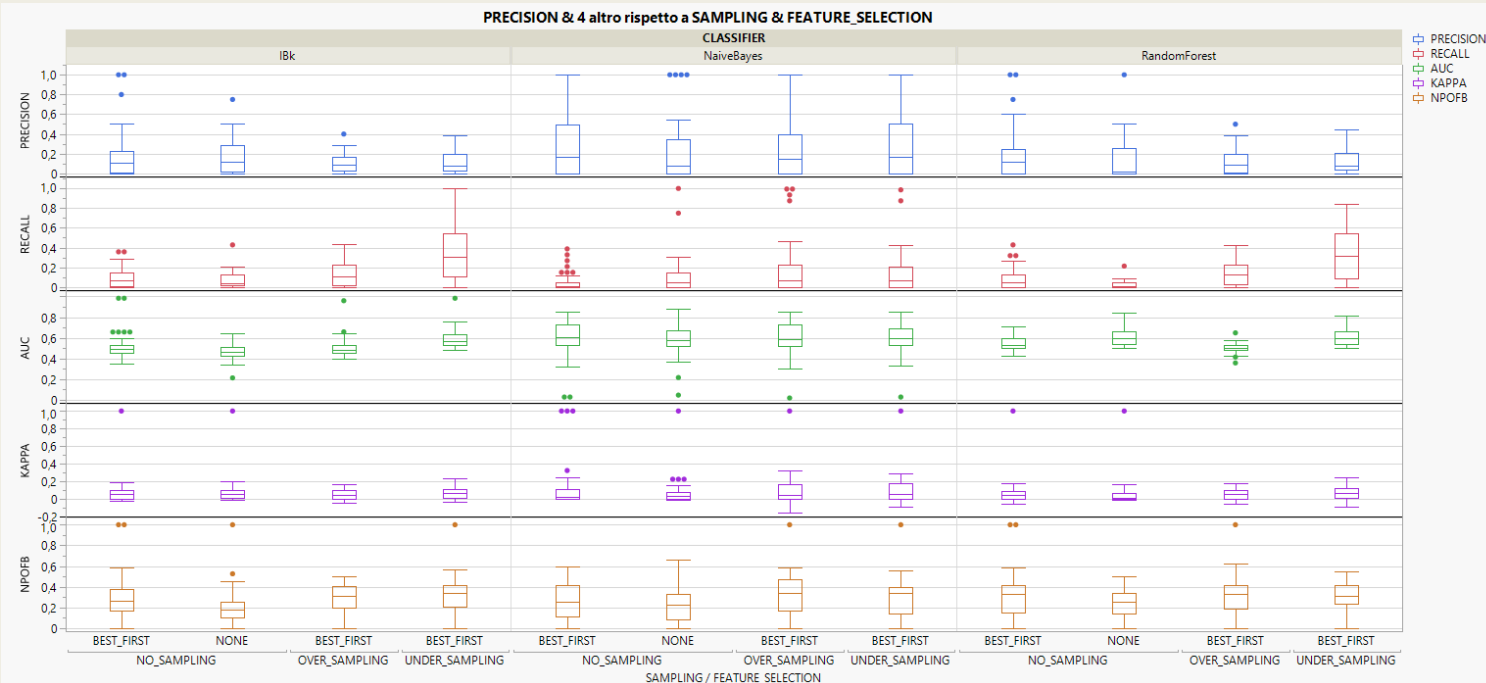
Risultati – Syncope - esecuzione con **feature selection**



- Nelle esecuzioni con solo **Feature Selection**, rispetto alle esecuzioni senza alcun filtro, troviamo:
 - **IBk**: valori medi di precision e recall sostanzialmente simili, maggiore variabilità della recall in caso di utilizzo di filtro
 - **Naive Bayes**: leggero miglioramento del valore medio di precision, valore di recall simile
 - **Random Forest**: anche in questo caso si nota un leggero miglioramento nella precision ma anche un miglioramento più accentuato nella recall.

Attualmente, **Random Forest** e **Naive Bayes** sembrano essere i classificatori **migliori**, con risultati molto simili.

Risultati – Syncope - esecuzione con feature selection e sampling

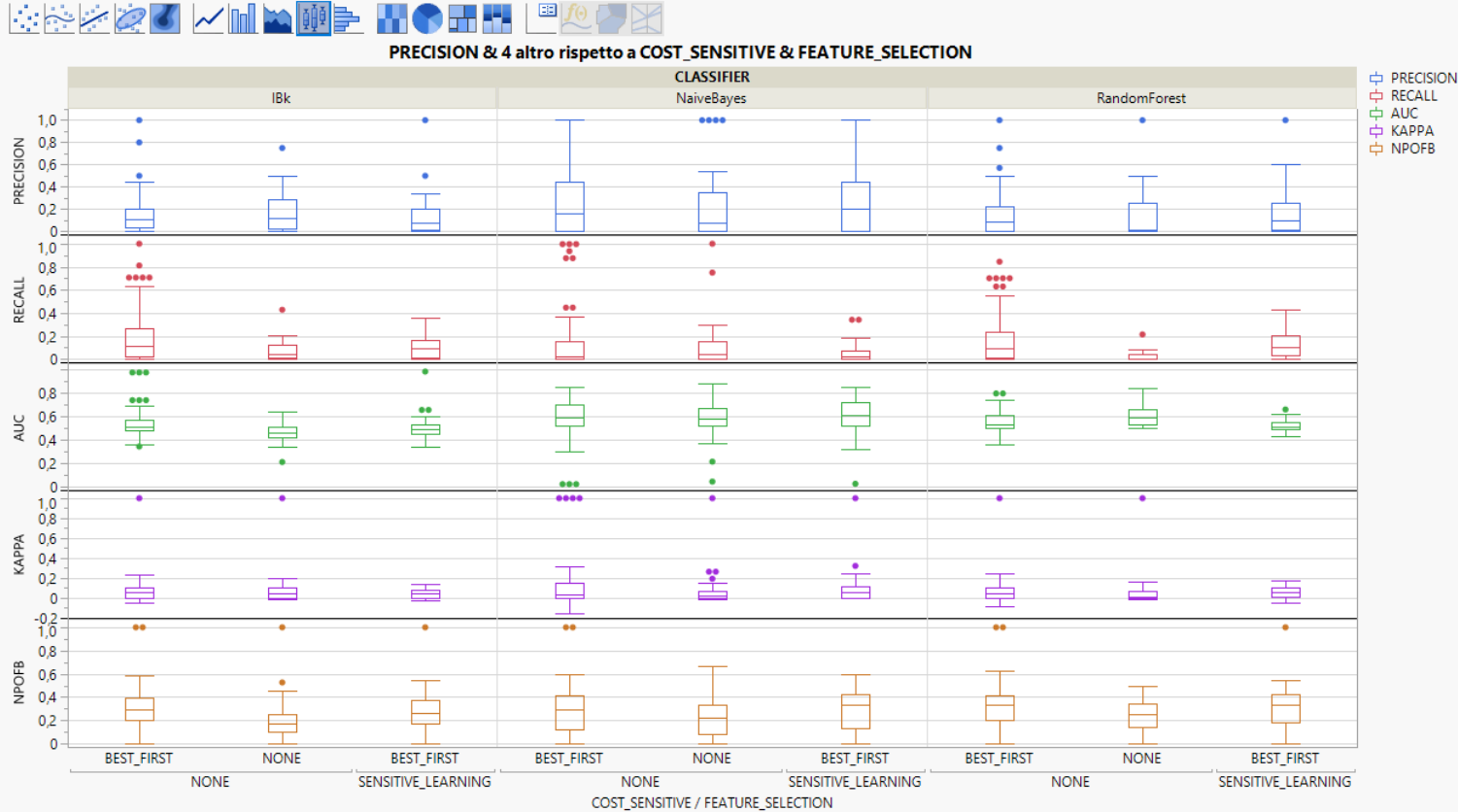


- Nelle esecuzioni con **Feature Selection** e **Over Sampling** si nota:
 - **IBk**: un valore di precision simile al caso con filtro, ma un aumento nel valore di recall, come atteso, dovuto all'aumentare della percentuale di positivi;
 - **Naive Bayes**: valore di precision molto simile rispetto al caso con filtro e un aumento nel valore di recall;
 - **Random Forest**: una leggera diminuzione nella precision rispetto al caso con filtro, ma un aumento nella recall.
- Nelle esecuzioni con **Feature Selection** e **Under Sampling** si nota:
 - **IBk**: un riduzione più marcata del valore di precision ed un aumento molto evidente del valore di recall, dovuto alla diminuzione della percentuale dei negativi;
 - **Naive Bayes**: valori molto simili al caso con oversampling, con precision e recall molto simili
 - **Random Forest**: un riduzione più marcata del valore di precision ed un aumento molto evidente del valore di recall, come atteso.

Anche per syncope, in caso di Under Sampling si osserva un generale aumento del valore di recall

In generale, in caso di Over Sampling, **Naive Bayes** risulta essere il classificatore migliore, per Under Sampling invece, **IBk** e **Random Forest**, con performance simili, sembrano essere i classificatori con le performance migliori.

Risultati – Syncope - esecuzione con **feature selection** e **sensitive learning**



- Nelle esecuzioni con **Feature Selection** e **Sensitive Learning** si nota:
 - **IBk**: valori stabili di precision e recall. Precision leggermente inferiore rispetto al caso senza nessun filtro.
 - **Naive Bayes**: leggero aumento nella precision, simile al caso di utilizzo di filtro, valore di recall, in media simile ai casi confrontati.
 - **Random Forest**: valore di precision simile a quello ottenuto con solo filtro. Aumento della recall rispetto al caso senza filtro, ed in linea con il valore ottenuto in caso di utilizzo del filtro.

In questo caso **Naive Bayes** sembra essere il classificatore **migliore** per la precision, e **Random Forest** per quanto riguarda il valore di recall.

Conclusioni

- In generale, non è possibile definire il classificatore migliore in **assoluto**. I dati variano molto dal dataset iniziale e alle varie tecniche applicate:
 - *Un candidato ad essere la configurazione migliore per **BookKeeper** risulta essere **Random Forest** nella configurazione **Feature Selection + Sensitive Learning** in quanto presenta i valori medi più alti di recall e precision;*
 - *Per **Syncope** invece risulta essere **Random Forest** nella configurazione **Feature Selection + Over Sampling**.*
- I risultati ottenuti confermano le aspettative teoriche, mostrando un comportamento prevedibile in base alle tecniche applicate.
- Nel caso di dataset di dimensioni ridotte, come BookKeeper, l'uso di tecniche di Under Sampling potrebbe causare uno sbilanciamento significativo e una perdita eccessiva di informazioni. Per mitigare questo problema, è preferibile adottare approcci alternativi, come l'Over Sampling o il Sensitive Learning..

Considerazioni sulla validità dei risultati

- L'operazione di Cold Start nel calcolo della Proportion si basa sull'assunzione che alcuni progetti Apache selezionati (come Avro e Zookeeper per BookKeeper, oppure Proton e OpenJPA per Syncope) non presentino somiglianze significative con i progetti oggetto di studio.
- I **ticket** che **non presentano commit** associati vengono esclusi dal calcolo, poiché non forniscono informazioni utili.
- Le **informazioni** su JIRA, come le date di apertura, risoluzione e le versioni affette o fixed, vengono considerate **vere a priori**.
- Se un ticket presenta una lista di Affected Version (AV), viene considerata come Injected Version (IV) la **prima AV nella lista**.

Link

- GITHUB:

- [*DiTotto/ISW2-bookkeeper*](#)

- SONARCLOUD:

- [*Summary - ISW2-bookkeeper in Di Totto Luca SonarQube Cloud*](#)